SNS REPORT ELEC088 24/25 REPORT

SN:24034726

ABSTRACT

This report presents the design and implementation of an LSTM neural network-based box office prediction system for the 2015 film Jurassic World. The methodology employed encompasses the utilisation of historical daily box office data, encompassing data preprocessing, sequence construction, model training, and the design of client/server architecture. The implementation of a simple Oracle chatbot, capable of predicting the box office trend for the following day based on user queries, is central to the study. The experimental results demonstrate the efficacy of the model in capturing box office trends, paving the way for further optimisation in the future.

***Index Terms*—** Box Office Prediction, LSTM, Time Series, Client/Server, Oracle Chatbot

1. INTRODUCTION

In the context of the continuous development of the film industry, the ability to predict box office performance has emerged as a critical metric for evaluating the commercial success of films. Conventional forecasting methodologies frequently employ statistical models; however, deep learning techniques, notably long short-term memory (LSTM) networks, have demonstrated remarkable proficiency in time series forecasting in recent times.

This report focuses on the construction of a prediction model using the daily box office data of Jurassic World (2015) and the implementation of an interactive query service through a simple client/server architecture. The structure of the report is as follows:

Section 2 describes the dataset and data sources;
Section 3 details the data preprocessing, model design and training, and client/server architecture;
Section 4 shows the experimental results and analyses;
Section 5 gives a summary and directions for future improvement;

2. DESCRIPTION OF DATA SETS

The dataset utilised in this report is the daily box office data of Jurassic World (2015) from Kaggle [1]. This dataset includes a lot of films, and since I was a one-person group, I only chose about 160 days of data for one of the films. The following data are included in the dataset:

Movie_Title: The title of the movie.

Date: The date corresponding to each record.

Daily: Daily box office revenue.

Theaters: Number of theaters screening the movie on that day.

Rank: The box office ranking for that day.

Additional features derived during preprocessing include:

Day_Since_Release: The number of days since the movie's release.

Log_Daily: The logarithmic transformation (using log1p) of daily revenue to mitigate the large variance in box office amounts.

3. METHODOLOGY

3.1 Data Preprocessing and Sequence Construction

The preprocessing code reads the CSV file, converts the Date column to datetime format, sorts the data, calculates the Day_Since_Release, and performs a logarithmic

transformation on the daily revenue. Then, a sliding window approach (with a window length of 7 days) is applied to construct the time series dataset. The features used for each time step are Log_Daily and Theaters, and the target is the Log_Daily value on the following day.

Key code:

```
# Data Preprocessing (from train_model_pytorch.py)
df = pd.read_csv('Jurassic World Movie Database.csv')

df['Date'] = pd.to_datetime(df['Date'])
df = df.sort_values('Date').reset_index(drop=True)
df['Day_Since_Release'] = (df['Date'] - df['Date'].min()).dt.days
df['Log_Daily'] = np.log1p(df['Daily'])
processed_df = df[['Day_Since_Release', 'Daily', 'Log_Daily', 'Theaters', 'Rank']]

# Sequence construction
data = processed_df[['Log_Daily', 'Theaters']].values

scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(data)
sequence_length = 7
X, y = [], []
for i in range(len(normalized_data) - sequence_length):
    X.append(normalized_data[i:i+sequence_length])
    y.append(normalized_data[i+sequence_length][0])
X = np.array(X)
y = np.array(y)
```

Explanation:

The log transformation helps to manage the wide range of box office figures.

The sliding window method reduces the total number of samples; for example, with 160 constructed samples, an 80/20 split results in around 32 samples in the test set, which explains why the prediction graph shows approximately 30 days.

3.2 Model Design and Training

Using PyTorch, an LSTM network is defined with one LSTM layer (64 hidden units) followed by a dense layer and output layer. The model is trained on the constructed sequences using MSE as the loss function. The training and validation losses are monitored to assess the model's performance.

Key code：

```
# PyTorch Model Definition (from train_model_pytorch.py)
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc1 = nn.Linear(hidden_size, 32)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(32, output_size)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = out[:, -1, :]  # use last time step's output
        out = self.fc1(out)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

Training logs are recorded and plotted as training and validation loss curves (see Section 4 for experimental results). After training, the model is saved as pytorch_model.pth.

3.3 Client/Server Architecture

In order to implement online prediction, the server code must first load the trained model and normaliser. Subsequently, the server listens on the designated port to

receive client requests. Upon receiving a request, the server generates a prediction using the most recent seven days of data and returns the result to the client after post-processing. The client then simply sends a query and displays the returned results.

Key code：

```
# server_pytorch.py (Simplified version)
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind(('localhost', 8080))
server_socket.listen(5)
# For details on processing requests, calling prepare_input(), model prediction, and inverse_transform_prediction(), please refer to the full code

# client_pytorch.py (Simplified version)
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('localhost', 8080))
query = input("Enter your query: ")
client_socket.send(query.encode('utf-8'))
response = client_socket.recv(1024).decode('utf-8')
print("Server response:", response)
client_socket.close()
```

Additionally, a simple system architecture diagram (see Appendix) illustrates the overall workflow.

4. EXPERIMENTAL RESULTS AND ANALYSIS

Training Process:

Figure.1 shows the training and validation loss over the epochs. Both loss decrease gradually, indicating that the model is learning the temporal features of the box office data. Note that due to the sliding window approach and an 80/20 train-test split, the test set comprises roughly 32 samples, which corresponds to about 30 days of predictions.

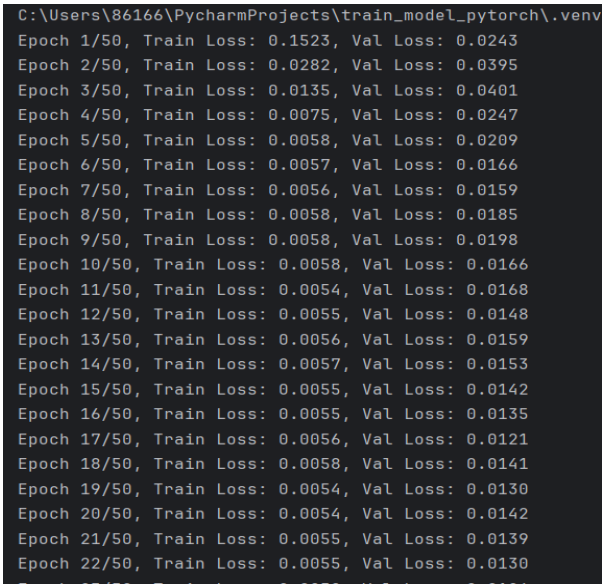

Figure 1: Training and Validation Loss

Prediction Results:

Figure.2 compares the predicted box office values with the actual values on the test set. The model is able to capture the general downward trend of the box office performance. However, significant differences exist at peak values, likely due to the influence of external factors (e.g., holidays, weather conditions) and the limited feature set used for prediction.

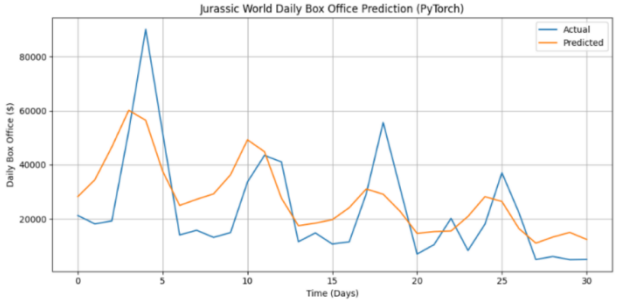

Figure 2. Comparison of projected results with actual data

Client Query Result:

Figure.3 shows the results obtained when the client sends a prediction request and gets the result.

```
Enter your query (e.g., 'Predict next day box office'): predict next day box office
Server response: Predicted next day box office: $12460.18
```

Figure 3. prediction results graphs in client

Discussion:

The limited number of test samples (≈30 days) restricts the range of the prediction graph.

The model's performance is affected by the inherent volatility of box office data and the limited features (only historical box office and number of theaters).

Future work could include additional features (e.g., audience ratings, competing film releases) and explore multi-step prediction strategies.

5. CONCLUSION

In this project, a daily box office prediction model for Jurassic World (2015) was constructed based on a LSTM network. An online query service for an Oracle chatbot was implemented through a simple client/server architecture. The experimental results demonstrate that the model has the capacity to capture box office trends and offer a novel approach to film box office prediction. Future work will focus on:

The incorporation of additional external factors (e.g., user ratings, social media data) into the model;The optimisation of the model structure and hyperparameters;The enhancement of the client interaction interface to improve system robustness and user experience.

6. REFERENCES

[1] L. LaRue, “Movie Attributes for 3400 Movies (2000 – 2020) [Dataset],” Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/lukelarue/movie-attributes-for-3400-movies-from-20002020 [Accessed: 27-Mar-2025].

GitHub: https://github.com/Yifei-Yuan/SNS-assignment.git

APPENDIX

A.System Architecture Diagram

Below is a textual version of the system architecture.